

AND JULIA EVANS
BY KATE SYLOR-MILLER



RECIPES FOR GETTING OUT OF A GIT MESS

DANGIT, GIT!

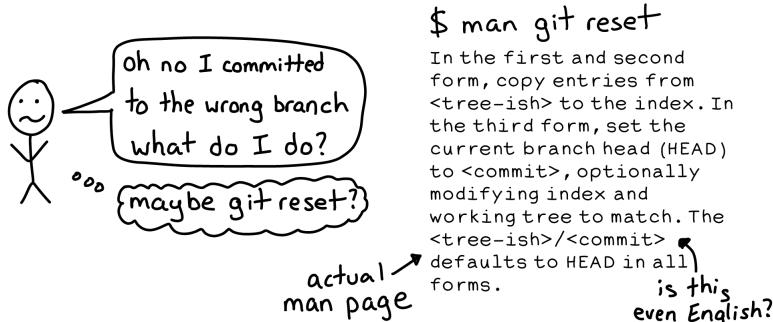
©

#\$%^&*

<https://dangitgit.com>
LOVE THIS?

What's this?

If you find git confusing, don't worry! You're not alone. People who've been using it every day for years still make mistakes and aren't sure how to fix them. A lot of git commands are confusingly named (why do you create new branches with `git checkout`?) and there are 20 million different ways to do everything.



This zine explains some git fundamentals in plain English, and how to fix a lot of common git mistakes.



By Katie Sylor-Miller & Julia Evans
Website: <https://ohshitgit.com>
Cover art by Deise Lino

 DANGIT!?! I did something terribly wrong,
 does git have a magic time machine?

Yes! It's called git reflog and it logs every single thing you do with git so that you can always go back.

Suppose you ran these git commands:

```
git checkout my-cool-branch ①  
git commit -am "add cool feature"②  
git rebase master③
```

Here's what `git reflog`'s output would look like. It shows the most recent actions first:

```
245fc8d HEAD@{2} rebase -i (start):③checkout master  
b623930 HEAD@{3} commit:          ②add cool feature  
01d7933 HEAD@{4} checkout:        ①moving from master  
:                           to my-cool-branch
```

If you really regret that rebase and want to go back, here's how:

```
git reset --hard b623930
```

```
git reset --hard HEAD@{3}
```

2 ways to refer
to that commit
before the
rebase

Table of Contents

* git fundamentals *

- 4 a SHA is always the same code
- 5 every commit has a parent commit
- 6 a branch is a reference to a commit
- 7 HEAD is the commit you have checked out
- 8 mistakes you can't fix
- 9 I need to change the message on my last commit!
- 10 I committed but I need to make one small change!
- 11-12 I accidentally committed to the wrong branch!
- 13 I tried to run a diff but nothing happened!
- 14 I have a merge conflict!
- 15 I committed a file that should be ignored!
- 16 I rebased and now I have 1,000 conflicts to fix!
- 17 I want to split my commit into 2 commits!
- 18 I want to undo something from 5 commits ago!
- 19 I did something terribly wrong, does git have a magic time machine?

DANGIT!! I want to undo something from 5 commits ago!

If you made a mistake but want to keep all of the commits since then, git revert is your friend!

git revert will create a reverse patch for the changes in a commit and add it as a new commit.

① Find the commit SHA for the commit you

want to undo

② Run:

git revert SHA

③ Enter a commit message for the revert commit

Now all of the changes you made in that commit are undone!

This is super useful if you push a bad commit to a shared repository and need to undo it!



A SHA always refers to the same code

Let's start with some fundamentals! If you understand the basics about how git works, it's WAY easier to fix mistakes. So let's explain what a git commit is!

Every git commit has an id like 3f29abcd233fa, also called a SHA ("Secure Hash Algorithm"). A SHA refers to both:

- the changes that were made in that commit see them with 'git show'
- a snapshot of the code after that commit was made

No matter how many weird things you do with git, checking out a SHA will always give you the exact same code. It's like saving your game so that you can go back if you die! You can check out a commit like this:

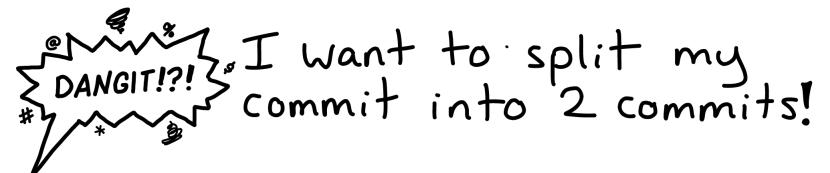
git checkout 3f29ab

SHAs are long
but you can just use the first 6 chars

This makes it way easier to recover from mistakes!



4



- ① Stash any uncommitted changes (so they don't get mixed up with the changes from the commit)

git stash

- ② Undo your most recent commit

git reset HEAD^

↑
safe: this points your branch at the parent commit but doesn't change any files

- ③ Use git add to pick and choose which files you want to commit and make your new commits!

- ④ Get your uncommitted changes back

git stash pop



You can use 'git add -p' if you want to commit some changes to a file but not others!

A branch is a pointer
to a commit

A branch in git is a pointer to a commit SHA
master → 2e9fab
awesomen-feature → 3bafeaf
fix-fypd → 9a9a9a

Here's some proof! In your favourite git repo, run
this command:

\$ cat .git/refs/heads/master
this is just a text file
with the commit SHA master
points at!

3 main ways to change the commit a branch points to:

Understanding what a branch is will make it WAY EASIER
to figure out how to get your branch to point at the right
commit again!
to fix your branches when they're broken: you just need
to fix your branches when they're broken: you just need

- * git commit will point the branch at the same commit as the remote branch
- * git pull will point the branch at the new commit
- * git reset COMMIT_SHA will point the branch at the new commit

16
branches with many conflicting commits, you can just merge
alternatively, if you have 2 commits, you can just merge

git rebase master
git rebase -i SHA_YOU_FOUND
④ Rebase on master
git merge-base
goes here
output of

git rebase all the commits in your branch together
③ Squash all the commits in your branch together

git merge-base master my-branch
from master
② Find the commit where your branch diverged
git rebase --abort

① Escape the rebase of doom
This can happen when you're rebasing many commits at once.

I started rebasing and
DANGIT!! Now I have 100000000
conflicts to fix!

HEAD is the commit you have checked out

In git you always have some commit checked out. HEAD is a pointer to that commit and you'll see HEAD used a lot in this zine. Like a branch, HEAD is just a text file.

Run `cat git/HEAD` to see the current HEAD.

Here are a couple of examples of how to use HEAD:

show the diff for the current commit:

```
git show HEAD
```

UNDO UNDO UNDO UNDO: reset branch to 16 commits ago !!

```
git reset --hard HEAD~16
```

↑
HEAD~16 means
16 commits ago

show what's changed since 6 commits ago:

```
git diff HEAD~6
```

squash a bunch of commits together

```
git rebase -i HEAD~8
```

↑
Rebasing a branch against itself 8 commits
ago lets you squash commits together!
(use "fixup")



I committed a file that should be ignored!

Did you accidentally commit a 1.5GB file along with the files you actually wanted to commit? We've all done it.

① Remove the file from Git's index

```
git rm --cached FILENAME
```

This is safe: it won't delete the file

② Amend your last commit

```
git commit --amend
```

③ (optional) Edit your `.gitignore` so it doesn't happen again



now your coworkers won't be stuck
downloading a HUGE git commit

7

Every commit (except the first one) has a parent commit
Suppose you had master checked out and ran
git merge feature-branch.
When that causes a merge conflict, you'll see
something like this in the files with conflicts:
<<<<< HEAD
if x == 0:
 return false
 if y == 6:
 return true
 else if x == 0:
 return true
 else if y == 6:
 code from master
 else:
 feature-branch
>>>>> D34367
if y == 6:
 if x == 0:
 code from master
 else:
 feature-branch
else:
 if x == 0:
 code from master
 else:
 feature-branch
HEAD
"make cats blue"
current+
commit+
parent
HEAD "add cats"
parent
HEAD "fix typo"
grand-parent
HEAD "initial commit"
⋮
b29aff

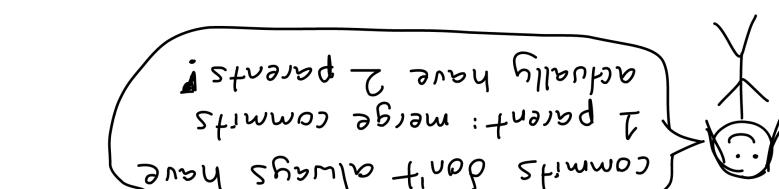
has a parent
every commit

commits don't always have
1 parent: merge commits
actually have 2 parents!

git checkout HEAD^

HEAD always refers to the current commit you have
checked out, and HEAD^ is its parent. So if you want to go
look at the code from the previous commit, you can run
git checkout HEAD^

git log shows you all the ancestors of the current
commit, all the way back to the initial commit



- 14
- ① Edit the files to fix the conflict
 - ② git add the fixed files
 - ③ git diff --check: check for more conflicts
 - ④ git commit when you're done → --continue if you're rebasing!
 - ⑤ Mel d (meldmerge.org) is a great choice!

To resolve the conflict:

```

<<<<< HEAD
if x == 0:
    return false
    if y == 6:
        return true
    else if x == 0:
        return true
    else if y == 6:
        code from master
    else:
        feature-branch
&gt;&gt;&gt;&gt; D34367
if y == 6:
    if x == 0:
        code from master
    else:
        feature-branch
else:
    if x == 0:
        code from master
    else:
        feature-branch

```

Something like this in the files with conflicts:
When that causes a merge conflict, you'll see
git merge feature-branch.

I have a
merge conflict?
DANGIT!!

mistakes you can't fix

Most mistakes you make with git can be fixed. If you've ever committed your code, you can get it back. That's what the rest of this zine is about!

Here are the dangerous git commands: the ones that throw away uncommitted work.

⚠ git reset --hard COMMIT

- ① Throws away uncommitted changes
- ② Points current branch at COMMIT

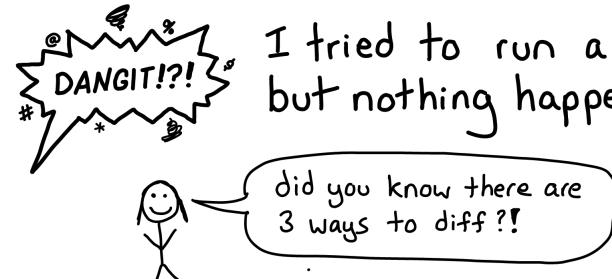
Very useful, but be careful to commit first if you don't want to lose your changes

⚠ git clean

Deletes files that aren't tracked by Git.

⚠ git checkout BRANCH FILE^{or directory}

Replaces FILE with the version from BRANCH.
Will overwrite uncommitted changes.



Suppose you've edited 2 files:

```
$ git status  
On branch master  
Changes to be committed: staged changes  
modified: staged.txt (added with 'git add')  
Changes not staged for commit:  
modified: unstaged.txt unstaged changes
```

Here are the 3 ways git can show you a diff for these changes:

- git diff: unstaged changes
- git diff --staged: staged changes
- git diff HEAD: staged+unstaged changes

A couple more diff tricks:

- git diff --stat gives you a summary of which files were changed & number of added/deleted lines
- git diff --check checks for merge conflict markers & whitespace errors

I committed something to my master branch.
I need to change the message on my last commit!
Then edit the commit message & save!
git commit --amend will replace the old commit
with a new commit with a new SHA, so you can
always go back to the old version if you really need
to.

git commit --amend

No problem! Just run:

DANGER!! I need to change the message on my last commit!
I committed something to my master branch.
Then edit the commit message & save!

① Make sure you have master checked out

git checkout master

② Create the new branch

git branch my-new-branch

③ Remove the unwanted commit from master

careful!
git reset --hard HEAD

④ Check out the new branch!

git checkout my-new-branch

git branch, and git checkout -b, both
create a new branch. The difference is
'git checkout -b', also checks out the branch



I committed but I need to make one small change!

- ① Make your change
- ② Add your files with git add
- ③ Run:

```
git commit --amend --no-edit
```



this usually happens to me when I forget to run tests/ linters before committing!

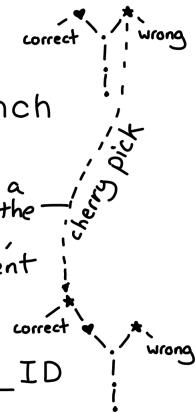
You can also add a new commit and use git rebase -i to squash them but this is about a million times faster.



I accidentally committed to the wrong branch!

- ① Check out the correct branch

```
git checkout correct-branch
```



- ② Add the commit you wanted to it

```
git cherry-pick COMMIT_ID
```

↑
use 'git log wrong-branch'
to find this

- ③ Delete the commit from the wrong branch

```
git checkout wrong-branch  
!git reset --hard HEAD^
```



be careful when running 'git reset --hard'!
I always run 'git status' first to
make sure there aren't uncommitted
changes and 'git stash' to save them
if there are